地球科学のF90

テキスト

熊本大学理学部地球科学科

目 次

第1章　フォートラン（Fortran90)によるプログラミング

1.1　プログラムのコンセプト

この章ではプログラムの実行のさせ方について実習する。最初に「ABC.F90」というファイルに以下の様な5行作ってみよう。これは、Fortran90 文法によって書かれた1つのプログラムなのである。

```
1 行目 program ABC
2 行目    real(8):: A
3 行目    A=1.0
4 行目    write(*,*) A
5 行目 end program ABC
```

プログラムは計算機内部での計算の命令手順を示したものである。プログラムの一般的なコンセプトを述べれば、

I. (特に断らない限り) 上の行から順番に命令を実行する。

II. プログラムを書く人間が適当に変数名を定義する。

III. 「何とか=何とか」というような一見等式にみえるようなものは、実は方程式でも何でもなく、「右辺の値を左辺の変数に代入する」という操作を示したステートメントである。

ということにつきる。

例えば、上のプログラムを例に構成を考えてみよう。I に従って、1行目から実行される。まず一行目の “program ABC” というのは、別に計算をするわけではなくて、このプログラム名が “ABC” であることを宣言している (このように宣言をする文を宣言文という)。2行めは “A” という名前の変数 (数を覚えておく場所) を使いますという宣言。変数には実数と整数 (その他沢山) があり、この場合は実数。次は3行目である。ここから、実際に実行される文となる。II に従って、“A” という名前の変数に、III のコンセプトに従って 1.0 なる数値を代入している。次4行目の “write(*,*) A” は変数 A をパソコンのディスプレー上に表示させよという命令である。従って、この文が実行された時点で、パソコンの画面に

```
1.00000
```

という変数 A の数値が表示されるはずである。最後の “end” はプログラムの末尾を表すステートメントで、Fortran のプログラムはすべて “end” で終わる約束になっている。

この例では各行が Fortran プログラムの意味の塊になっていた。これを、Fortran の文と呼ぶ。Fortran90 では一行に複数の文を “;” で区切って書くことができる。また、行の最後に “&” を付けると一つの文を複数の行にまたがらせることができる。

原始プログラム *(souce program)* **から目的プログラム** *(object program)* **を作り、実行させる方法。**

1. スタート → プログラム → Fujitsu Fortran 90 & C → 開発マネージャ
2. エディタ ボタンを押す
3. 原始プログラム入力
4. 名前 (A:xxxxxxxx.F90) を付けて保存
5. 開発マネージャに戻り、今保存した原始プログラム (A:xxxxxxxx.F90) を選ぶ
6. 翻訳 ボタンを押す
7. 開発マネージャで、今作られた目的プログラム (A:xxxxxxxx.EXE) を選ぶ
8. 実行 ボタンを押す

1.1.1　練習問題

例題 2〜4 のプログラムをエディターで作成し、Fortran で実行させよ。

《例題 2》　ファイル名は ABC2.F90 とせよ。

```
program ABC2
   implicit none
   real(8):: A,B
   A=1.0
   write(*,*) A
   read(*,*) B
   write(*,*) B
end program ABC2
```

《例題 3》 ファイル名は ABC3.F90 とせよ。

```
progrem ABC3
   implicit none
   real(8):: A
   integer:: I
   A=0.0
   do I=1,100
      A=A+20.0
   end do
   write(*,*) A
end program ABC3
```

《例題 4》 ファイル名は ABC4.F90 とせよ。

```
program ABC4
   implicit none
   real(8),dimension(5):: A
   integer::I
   A(1)=23.0
   A(2)=3.14
   A(3)=4.0
   A(4)=99.99
   A(5)=13.2
   do I=1,5
      write(*,*) A(I)
   end do
!
   do I=1,5
      A(I)=A(I)+0.01
   end do
   do I=1,5
      write(*,*) A(I)
   end do
end program ABC4
```

1.2 Fortran90 によるプログラミングの基本

これまでの実習でプログラムの見本さえ手元にあれば、一応、ファイルとして作ることとと走らせることはできるようになった (はずである)。これ以降では、Fortran90 言語を用いた科学技術計

算のための基本的な事柄について解説と演習を行ない、プログラミングの基礎を身につけることを目指したい。

1.2.1　プログラムのコンセプトについてもう一度

I. プログラムは上の行から順番に命令を実行する。

II. 変数名はプログラムを書く人間が適当に定義すればよい。

III. 「何とか=何とか」というような一見等式にみえるようなものは、**「右辺の値を左辺の変数に代入する」という操作**を示している。

ということをもう一度銘記してほしい。

Fortran の文法規則について一から説明していては短期間の実習ではとうてい時間がないし、またそれは必要ないことでもある。重要なことは、要点を要領よく知ることである。教員サイドとしては、実際の地球科学の研究のさまざまな局面で経験的に知ったコツのようなものを伝授したいと思う。それらは、Fortran90 の文法の使い方の一部に過ぎなかったり、書き方の 1 つの流儀に過ぎなかったりするかもしれないが、ゴールへの最短距離であると思う。Fortran プログラミングの基本は以下の通り。

I. プログラムの第 1 行目は、プログラム文を書くこと (プログラムファイル名として、例えば「XXX.F90」に対しては、1 行目で “`program　XXX`” と指定すること (例題 1〜4 参照))。

> もしこれを指定しないと (つまり、プログラム文を入れ忘れると)、オブジェクトモジュールとして、FTMAIN.OBJ　ができるので、リンクの際には、ロードモジュールとして XXX.EXE を作るのであれば、 LINK FTMAIN,XXX,,, としなければならない。

II. プログラム文の次 (すなわち 2 行目) に、

```
implicit none
```

なるステートメントを入れること。こうしておくと宣言していない変数を使った時にコンパイラが指摘してくれる。

III. ほとんどすべての科学技術計算は、倍精度 (有効数字 16 桁) で行うのが普通である。実数の宣言には

```
real(8):: A,B
```

などと、`real(8)` を用いる。また、整数の宣言は

```
integer:: I,J
```

などとする。Fortran の慣行として、頭文字が A から H, O から Z までの変数を実数に、頭文字が I，J，K，L，M，N の変数を整数にすることが多い。Fortran90 では従う必要はないが、不都合のない時にはそう使い分けることを勧める。

このような、倍精度の指定をしないと、単精度 (有効数字 8 桁) の計算になってしまう (電卓より精度が低い！)

IV. **プログラムの最後は** end **文で終わること。**

V. 変数は、実数・整数の区別を厳密に行うこと。

- 実数型の場合、「A=1.0」　または「A=1.0D0」(1.0×10^0 の意)
- 整数型の場合、I=1　(決して　I=1.0　とはしない) のように書くこと。
- 整数→実数の変換は、例えば、A=dble(I) のように、dble(実数化:　倍精度) を使って行う。
- 実数→整数の変換は、I=int(A) のように、int(整数化:　整数部分だけを取り出す、小数部は切捨て) を使って行う。

例)I=3 のとき、A=dble(I) とすれば、A=3.0 のようになり、A=2.3 のとき、J=int(A) とすれば、J=2 のように小数部が切り捨てられ、整数部のみが表示される。

VI. どの行でもエクスクラメーションマーク (!) 以降は無視される。これを用いてプログラムにコメントを入れることができる。

VII. 最後にアンパサンド (&) を入れると、次の行はその行の続きとなる。

1.3　入出力文 (cf. 例題 1、2)

read(*,*)　　パソコンキーボードより入力
write(*,*)　パソコンディスプレー (画面) に出力
(二つ目の*は、書式の指定をしないことを意味する)

書式とは、例えば「1.0」という数値を表示させる際に、
「1.00」　　と表示させるか
「1.0000」　　と表示させるか
「1.0E0」　　と表示させるか、などの違いをいう。
書式を指定するには、FORMAT を使う (後述)。

例題 2 をもう一度見てみよう。

```
program ABC2
   implicit none
   real(8):: A, B
   A=1.0
   write(*,*) A
   read(*,*) B
   write(*,*) B
end program ABC2
```

3 行目で変数 A, B が定義される。

4 行目で 1.0 なる値が A に代入されている。

5 行目で A の値がパソコンの画面に出力される。

6 行目では変数 B の値をキーボードから入力する。

プログラムを走らせると 5 行目まで実行された時点でパソコン側はキーボードからの入力待ちになっているので、適当な実数の値をキーボードから入れてやる。例えば、「987.654321」と入れたとしよう。すると 6 行目の「write(*,*) B」によってこの値がディスプレーに表示される。7 行目でプログラム実行終了となる。

1.4　四則演算、組み込み関数

四則演算は「+」「-」「*」「/」の記号を用いて表現する。

A の B 乗は「A**B」と表す。

「A*B/C*D」は、((A*B)/C)*D　のことである (つまり数式の演算順序と同じ)。あやふやなときはためらわずに括弧を使うこと (間違ったプログラムを書くよりはるかにまし。括弧を使ってもエラーはでない)。

A の平方根は「SQRT(A)」と表現する。

A の正弦は「SIN(A)」と表現する。このように「関数名 (変数名)」のように表せるものを組み込み関数という。前に出てきた、dble(…)、int(…) も組込み関数の 1 つである。Fortran で使用できる組み込み関数の例を次に示す。

組込み関数一覧表

数値関数

abs(a)	絶対値
dim(a,b)	if (a>b) then a-b else 0
sign(a,b)	a の絶対値に b の符合を付ける
aint(x)	切り捨て (実数を返す)
int(x)	切り捨て (整数を返す)
anint(x)	四捨五入 (実数を返す)

nint(x)	四捨五入（整数を返す）
ceeling(x)	引き数以上で最小の整数（整数を返す）
real(i)	実数型への変換
dble(i)	倍精度実数型への変換
mod(a,b)	余り
modulo(a,b)	余剰関数

数学関数

sin(x)	正弦関数（ラジアン）
cos(x)	余弦関数（ラジアン）
tan(x)	正接関数（ラジアン）
asin(x)	逆正弦関数（ラジアン）
acos(x)	逆余弦関数（ラジアン）
atan(x)	逆正接関数（ラジアン）
atan2(x,y)	逆正接関数（ラジアン）
exp(x)	指数関数
log(x)	自然対数
log10(x)	常用対数
sinh(x)	双曲線正弦関数
cosh(x)	双曲線余弦関数
tanh(x)	双曲線正接関数
sqrt(x)	平方根

引き数の x,y は実数のみを i は整数のみを表す。a,b は実数、整数のいずれでも入ることを示す。かえってくる変数の型は特に指定のない限り引き数と同じになる。

例)

```
program XYZ
   implicit none
   real(8):: A,B,C,D,E,F
   A=1.0D0
   B=4.0D0
   C=A+B+3.0D1
   D=SQRT(C)
   E=D**2-(A+B)
   F=SIN(E)-2.0**(1.0/3.0)
   F=F-4.0
   write(*,*) F
end program XYZ
```

これは結局、

$$\sin((\sqrt{1+4+30})^2-(1+4))-2^{\frac{1}{3}}-4$$

のような演算を行っているわけである。

「F=F-4.0」というのは奇妙に思うかもしれないが、右辺の値、つまり、その前の行までの F の値 (SIN(E)-2.0**(1.0/3.0)))-4.0) を左辺の変数 F の値としてに新たに定義するというコンセプトを思い起こせば理解できよう。もちろんプログラムの最後の 3 行を、

```
   G=F-4.0
   write(*,*) G
end program XYZ
```

のように書いても同じ結果が得られるのはいうまでもない。(ただし、この場合は 3 行目の宣言に G も入れないといけない。)

※問題 1　整数 x_1, x_2, x_3 を読み込んで、

$$y=13.5x_1+4.8x_2+\frac{5.2}{x_3}$$

を求め、その整数部を出力させよ。プログラム名は MON1.F90 とせよ。

(プログラムの中で、問題の式と同じ変数名をプログラム中に定義しなければならない理由はない。)

※問題 2　2 次方程式

$$ax^2+bx+c=0$$

の解を求めるプログラムを書け。プログラム名は MON2.F90 とせよ。

(方程式の各次の係数 a, b, c をキーボードから入力すれば、解の公式に従って、2 つの解がディスプレーに出力されるようなプログラムを作ればよい。)

1.5　整数と実数の区別について再度注意

繰り返して言うが、Fortran では変数の

整数型　(小数点がない)
実数型　(小数点がある)
文字型　(英語の文字)
など

変数のタイプの区別を厳密に行う。

初心者は、1 つのステートメント (文) に整数型と実数型の変数が混在するような書き方はしないようにせよ。

1 つのステートメントに整数型と実数型が混在してよいのは、

整数型→実数型　の変換： e.g., B=dble(I)
実数型→整数型　の変換： e.g., I=int(B)

のように、dble **や** int **を含む文だけしかないと心得よ。**

例えば、上のような IMPLICIT 宣言文をしたのなら、

```
M=7
N=2
A=M/N
```

のようなプログラムは書かないこと。なぜなら 3 行目は左辺が実数型、右辺に整数型が入っているからでこのような混在はよくない。

この計算の結果、A=3.5 という値が得られると思うのは間違いである。A=3.0 になっているはずである。なぜなら、M ／ N は 7 ／ 2 の整数部だけを取り出すことになるので (3.5 → 3、つまり小数部分切捨て)、3 になるのである。A は、この 3 という整数を実数型として扱うことになるので 3.0 になるわけである。このようなことをいちいち考えるのは慣れるまではいささかややこしい。また、整数型と実数型を混在させたステートメントを書くと、今の例のように、初心者は実際の値 (3.0) とは違う値 (3.5) がでてくると期待しがちである。

従って、

* **実数型だけの文を書く。**
* **整数型だけの文を書く。**

というふうにプログラムを書いて欲しい。

上の例でいえば、

```
M=7
N=2
PM=dble(M)
PN=dble(N)
A=PM/PN
```

のように、整数 M，N をあらかじめ “real( )” を使って実数化し、変数 A(実数型) が現れる文は、PN，PM(実数型) を使って書くようにする。

また、整数型の変数が現れるステートメントは整数型の変数だけを使って書くこと。

例えば、

```
J=M-(M/2)*2
```

のようなステートメントがそうである。

例えば、M=31 のとき (M/2) は 15.5 の整数部分 15 である。従って、(M/2)*2 は 30 となるので、右辺の値は 1、すなわち J=1 となる (このステートメントは、問題 7 に使える)。

整数→実数　の変換は　real
実数→整数　の変換は　int

を使って行うこと。

同様に、数字についても、実数型・整数型の区別をして欲しい。

数字　2　　は　整数型
数字　2.0　や　2.0D0　は　実数型

である。実数型の A、整数型の J に対して、A=2 と書いたり、J=3.0 と書いたりしないように (A=2.0, J=3 と書くべし)。

従って、例えば問題 2 のプログラムの中で、2 次方程式の根の公式を使う部分では、

```
X1=(-B+SQRT(B**2-4*A*C))/(2*A)
```

と書くのではなく、

```
X1=(-B ＋ SQRT(B**2-4.0*A*C))/(2.0*A)
```

と書いて欲しい (2 乗の 2 は整数の方がよい)。

同様に、上の例でも整数型の計算部分に、

```
J=M-(M/2.0)*2.0
```

というようなマネはしないこと。

組み込み関数 dble と int を使えば、いま述べたような書き方は守れるはずである。

1.6　do 文……繰り返し計算の方法 (cf. 例題 3、4)

「計算の繰り返し」はコンピューターの最も得意とする演算形態である。ある計算の繰り返しをさせるためには、その計算を記述したステートメントを「do 文、end do 文」で挟んでやればよい。例を示す。

```
do I=1, 100
：                    この部分が
：                    繰り返される
end do
```

「do I=1, 100」と「end do」で挟まれた部分の計算が、100 回繰り返される。そのとき、繰り返しの 1 回目は変数 I が 1, 2 回目は 2, 100 回目は 100 となり、I が 1 から 100 まで繰り返しの度に 1 つずつ増えてゆく。

例題 3 をもう一度見てみよう。

```
program ABC3
   implicit none
   real(8):: A
   integer:: I
   A=0.0
   do I=1,100
      A=A+20.0
   end do
   write(*,*) A
end program ABC3
```

「A=A+20.0」という、右辺の値を新たに左辺の値 (すなわち A) とせよ、という意味の演算が、100 回繰り返されることになる。

```
   do I=1,100
      A=A ＋ 20.0
   end do
```

の部分を、do 文を使わずに書き直せば、

```
   A=A+20.0          ……このとき I=1
   A=A+20.0          ……このとき I=2
      :                        :              100 回
      :                        :
   A=A+20.0          ……このとき I=100
```

のようになる。こう書いたことと同じことをやっているのである。

この do—end do の計算の結果、A の値は 1 回繰り返す毎に 20.0 ずつ加算されて、最終的に 2000.0 になる。

ちなみに do—end do は、BASIC 言語では、「FOR-NEXT 文」に対応する。このように do—end do 文は、コンピューターの最も得意とする、繰り返し演算を行うためにある。これを使えば、Σ や Π のような計算が簡単にできることが察せられるであろう。

【注意】 一般に、標準の Fortran では、変数の初期値は、必ずしもゼロではない。従って、Σ や Π を do—end do 文を使って求めるときは、do 文より前に、変数の値を初期化しておくこと (例題 3 の 3 行目のことである。これを入らなければ変数 A は繰り返しの前にいったいいくらなのか設定されていない)。

くどいようだがもう少し例をあげよう。

例 1.

```
   do I=1,3
      read(*,*) Q
```

```
   write(*,*) Q
end do
```

上の do ループは、

```
read(*,*) Q
write(*,*) Q              ←  I=1   (繰り返し 1 回目)
read(*,*) Q
write(*,*) Q              ←  I=2   (繰り返し 2 回目)
read(*,*) Q
write(*,*) Q              ←  I=3   (繰り返し 3 回目)
```

と書きかえても同じである。要するに、do 文と end do 文ではさまれている部分が 3 回 (I=1,3) 繰り返されているわけである。この例では数値 Q をキーボードから読ませてディスプレーに出力する操作が 3 回繰り返されることになる。

例 2.

```
M=0
do K=1,5
   M=M+K
end do
```

これは、書き換えると、

```
M=0
M=M+1                ←  K=1     左辺の M は  M=0 + 1
M=M+2                ←  K=2     左辺の M は  M=1 + 2
M=M+3                ←  K=3     左辺の M は  M=3 + 3
M=M+4                ←  K=4     左辺の M は  M=6 + 4
M=M+5                ←  K=5     左辺の M は  M=10 + 5
```

と同じである。この結果、$1 \times 2 \times 3 \times 4 \times 5 (= \sum_{k=1}^{5} k)$ という級数が求められたことがわかる。

例 3.

```
L=1
do J=1,4
   L=L*J
end do
```

上のようなループは、

```
L=1
L=L*1                ←  J=1     左辺の L は  L=1*1
```

`L=L*2`	←	`J=2`	左辺の L は `L=1*2`
`L=L*3`	←	`J=3`	左辺の L は `L=2*3`
`L=L*4`	←	`J=4`	左辺の L は `L=6*4`

と同じである。この結果、$1 \times 2 \times 3 \times 4 (= \prod_{j=1}^{4} j)$ という乗積が求められることがわかる。

このように do **ループを使えば、同じ作業の繰り返しをしたり、いろいろな級数 ($\sum$) や乗積 ($\prod$) を求めることができる。**

※問題 3　A=1，2，…，20 に対して、その平方根を求め出力せよ。(プログラム名 MON3.F90)

※問題 4　1 から N までの自然数の和 $\sum_{i=1}^{n} i$，2 乗和 $\sum_{i=1}^{n} i^2$ をもとめるプログラムを作れ。(プログラム名 MON4.F90)

※問題 5　10! を求めるプログラムを作れ。(プログラム名 MON5.F90)

※問題 6　ある実数 x を読み込み、$x^{\frac{1}{2n}}$ を $n = 1, 2, \cdots, 10$ に対して順に出力するプログラムを作れ。(プログラム名　MON6.F90)

1.7 配列変数 (cf. 例題 4)

普通、変数 A，B といえば、それぞれ A=2.3，B=4.5 のような 1 つの値を持つわけであるが、配列変数とは、はじめに例えば、“real(8):: A(5)” と宣言することによって、A に 5 つの添え字をつけることができる。

意味がよくわからないかもしれないが、例題 4 を見てみよう。

```
program ABC4
   implicit none
   real(8),dimension(5):: A
   integer:: I
   A(1)=23.0
   A(2)=3.0
   A(3)=4.0
   A(4)=99.99
   A(5)=13.2
   do I=1,5
      write(*,*) A(I)
   end do
!
   do I=1,5
      A(I)=A(I)+0.01
   end do
   do I=1,5
      write(*,*) A(I)
```

```
   end do
end program ABC4
```

ここで、A(1),A(2),…, A(5) はそれぞれ全く別の変数と考えてよい。つまり、

```
A1=23.0
A2=3.0
    :
    :
A5=13.2
```

と書いても同じことである (単に添え字を 5 つつけて 5 つの変数を定義しただけ!)。

配列変数は、例えば、多量のデータを処理する場合 (これもコンピューターのオハコ) などに使われる。観測量 Y のデータが例えば 1000 点あるとき、1000 個もの変数名をそれぞれ与えることは実際上できない。このようなとき、

```
real(8),dimension(1000):: Y
```

とすれば 1000 個のデータに対して、(添え字付き) 変数を割り振ることができ、便利である。

くどいようだがもう一度説明する。例えば、プログラムの冒頭、IMPLICIT 文の次に

「`real(8),dimension(5):: A,B`」

と宣言することによって、変数 A, B はそれぞれ 5 つの成分をもつ配列変数となる。配列変数とは成分を持つ変数のことである。いまの宣言をすることによって、

A(1),A(2),…,A(5)

B(1),B(2),…,B(5)　のように 10 個の変数を使うことになったと考える。つまり、A，B には実は添え字が付いていて、A1，A2,…，A5，B1，B2,…，B5 のようになっていると考える。だから当然、例えば A1 と A2 は違う値を持っている。下のプログラムを参照されたい。

```
program EXAM
   implicit none
   real(8):: A(5)
   A(1)=2.3
   A(2)=3.4
   A(3)=4.3
   A(4)=7.7
   A(5)=9.2
   write(*,*) A(1)
   write(*,*) A(2)
   write(*,*) A(3)
   write(*,*) A(4)
   write(*,*) A(5)
end program EXAM
```

この例では、A(1)〜A(5) の値がはじめに与えられていて、それが WRITE 文でディスプレーに出力されるわけである。この例でわかるように、A(1), A(2),…, A(5) は全く別の数値を持つ別の変数と考えられる。従って、上のプログラムを書く代わりに、配列変数を使わないで、

```
program EXAM2
    implicit none
    real(8)::P,Q,R,S,T
    P=2.3
    Q=3.4
    R=4.3
    S=7.7
    T=9.2
    write(*,*) P
    write(*,*) Q
    write(*,*) R
    write(*,*) S
    write(*,*) T
end program EXAM2
```

と書いても同じである。では、なぜあえて配列変数を書くかというと、一番上のプログラム EXAM は、実は

```
program EXAM3
   implicit none
   real(8),dimension(5):: A=(/2.3,3.4,4.3,7.7,9.2/)
   integer:: I
   do I=1,5
      write(*,*) A(I)
   end do
end program EXAM3
```

ともっと短く書き換えることができるのである。こちらのプログラムの方がはるかに簡単だ! 例えば、100 個もの多数のデータを読み込ませたり計算させたり出力させたりするとき、変数の名前を 100 個もいちいち名づけるのは不便である。そこで、配列変数を使うわけである。

実は配列変数は do 文と併用したときに最も威力を発揮する。例えば、

```
do I=1,5
   write(*,*) B(I)
end do
```

上の do ループは、

```
write(*,*) B(1)          ←  I=1   (繰り返し 1 回目)
write(*,*) B(2)          ←  I=2   (繰り返し 2 回目)
```

```
write(*,*) B(3)          ←  I=3   (繰り返し 3 回目)
write(*,*) B(4)          ←  I=4   (繰り返し 4 回目)
write(*,*) B(5)          ←  I=5   (繰り返し 5 回目)
```

と同じである。やはり do—end do 文ではさまれている部分が 5 回繰り返されているが、I の値が、1 回目は I=1、2 回目は I=2、というように、5 まで 1 ずつ増えてゆくことがわかることと思う (それが、I=1,5 の意味なのだ！)。

このような理解の上で、例題 4 をもう一度見てほしい。

※問題 7　15 個の実数を順にキーボードから読み込み、その総和を出力するプログラムを作れ。(プログラム名 MON7.F90)

1.8　条件文 (if)……プログラムの分岐

例 1.

```
program REI1
   implicit none
   integer:: I,J
   read(*,*) I
   J=0
   if(I > 100) J=101
   if(I <= 50) J=1
   write(*,*) J
end program REI1
```

if 文は、条件を判断するための文である。上の例では、I>100 ならば J=101 となり、I=50 ならば J=1 となる。つまり、if(……) の条件 (…の部分のことを論理式という) が満たされる場合には、実行文 (上の例では、J=101　や J=1 のこと) が実行される。

I. if 文はこのように、論理式と一緒に使う。

$>$	.GT.	>
$\geqq$	.GE.	>=
$<$	.LT.	<
$\leqq$	.LE.	<=
$=$	.EQ.	==
$\neq$	.NE.	/=
論理和	.OR.	
論理積	.AND.	
否定	.NOT.	

II. if(A.NE.B) や if(A/=B) のような形に書くこと。ただし、A と B は同じ型 (実数型か整数型)

であり、実数型の時は、GT と LT しか意味がない。(if(A = B) は if(ABS(A-B) < 1.E-8) の様に書く。どうしてか考えてみよう)

※問題 8　整数を読み込み、それが偶数であるか奇数であるかを判断するプログラムを考えよ (偶数なら 100 を、奇数なら 999 を画面に表示させよ)。　(プログラム名 MON8.F90)

例 2.

```
program REI2
   implicit none
   integer:: III,J,I
   read(*,*) III
   J=0
   I=0
   if(III>100) then
      J=101
      I=201
   end if
   write(*,*) J,I
end program REI2
```

例 2 のような場合は、論理式が満足されるときに、if(…) then　と end if で挟まれた部分が実行されることになる。1 つの if に対して、実行文が　2 行以上存在するときは、このような形に書けばよい。これをブロック if 文という。

III. ブロック if 文の形式は、

```
if(……) then
     ……
     ……
else
     ……
     ……
end if
```

である。 上記では、論理式が満足されない時に、else と end if で挟まれた部分が実行される場合も含めて書いてある。それぞれの “……” は空文であっても構わない。

1.9　初期値宣言文

つまらないことであるが、変数の初期値は初期値宣言文を使っても書くことができる。例えば、前節の例 2 の場合、4 行目、5 行目の「J=0」「I=0」をステートメントとして書くかわりに

```
program REI2
   implicit none
   integer:: J=0,I=0
   integer:: III
   read(*,*) III
   if(III.GT.100) then
      J=101
      I=201
   end if
   write(*,*) J,I
end program REI2
```

と書き換えができる。

I. 初期値の宣言は、

```
real(8):: ABC=E
```

の形式で行う (これは変数 ABC の初期値が E の場合)。

配列変数に対しては、例えば 1000 次元の変数 A(1000) に対しては、すべての要素を 0.0 に設定する場合は、

```
real(8),dimension(1000):: A=0.0
```

と書けばよい。

1.10　GO TO 文

GO TO　文は飛び越しを命令する。

例 3.

```
program REI3
   implicit none
   real(8):: A,B
   integer:: III
   read(*,*) A
```

```
   if(A/=0.0) goto 100
      III=9999
   write(*,*) III
   stop
100  B=1234.0/A
   write(*,*) B
end program REI3
```

例 3 では、0.0 を読み込ませた場合、ゼロで割り算はできないので、9999 を表示させて終了する仕組みになっている。

※問題 9　問題 1 のプログラムで、X3=0 を入力してもエラーがでないようにプログラムを修正せよ (X3=0 のときは割り算を実行しないようにせよ)。　(プログラム名 MON9.F90)

1.11　繰り返し 2 (do の他の使い方)

プログラミングも少し分かって来ただろうから少しずつ一般的な場合に付いて述べる。

`do i=1,100` の `i` を制御変数、`i=1,100` の部分を制御式と呼ぶ。do 文の制御式は `i=1,100,2` と書くことも出来て、この場合は 1,3,5....99 と 1 から 2 おきに変化する。`i=1,100` は `i=1,100,1` の省略形だったのである。`i=100,1,-1` も可能である。`i=100,1` と書くと do ブロックは一度も実行されない。

また、do 文の制御式は省略することもできる。その場合は無限ループを表す。このループから脱出するには exit 文を用いる。また、do ブロックの残りの部分を無視して、頭に戻る cycle 文がある。これらを使ったプログラムの例を以下に示す。このプログラムは 0 が入力されるまでに入力された数と全部と正の数のかずをかぞえて出力するものである。

```
program counting
   implicit none
   integer:: num,total=0,count=0
   getData: do
      write(*,*) 'Input number '
      read(*,*) num
      if( num == 0 ) exit getData
      total = total+1
      if( num<0) cycle getData
      count=count+1
   end do getData
   write(*,*) 'Nyuryoku sita su no kazu=',total, &
              ' Sei no su no kazu=',count
end program counting
```

1.12　配列２

前に配列の説明をした時には 100 個の変数と言う意味で

```
real(8),dimension(100)::a
```

などと宣言した。この添字の範囲は負の数にした方が都合が良いことが有る。例えば

```
real(8),dimension(-50:50)::a
```

という宣言をすると a(i) の i は -50 から 50 の範囲を取ることができる。また、配列の次元を 2 次元 3 次元と増やすこともできる。

```
real(8),dimension(-50:50,-50:50,-50:50)::a
```

とすると、3 次元の -50 から 50 までの格子点の値を格納することができる。

Fortran90 では、配列の各成分どうしの演算をまとめて書くことができる。

```
real(8),dimension(3)::a,b,c
 :
c(:)=a(:)+b(:)
```

で、ベクトル a とベクトル b の和をベクトル c とするという計算ができる。

```
do i=1,3
   c(i)=a(i)+b(i)
end do
```

を一行で書くことができるのである。また、配列どうしの演算にスカラーをまぜることができる。ベクトル a をスカラー k 倍するには

```
real(8),dimension(3)::a,b
real(8)::k
 :
b(:)=k*a(:)
```

と書けばよい。配列どうしの演算は、配列各要素の演算をまとめて書いたものなのだから、一つの式に登場する配列の大きさは揃っているなければならない。

配列は全部を参照せずに部分だけ参照することができる。これを上手に用いると、異なるサイズの配列の計算に配列式を利用できる。例えば

```
a(2:4)      a(2),a(3),a(4)
a(:,3)      a(1,3),a(2,3),a(3,3)
a(1:9:3)    a(1),a(4),a(7)
```

はいずれも同じサイズの配列で一つの配列式で用いることができる。

配列の定数は

```
(/2,3,5/)
```

などと書く。これを使って、

```
a(/3,1,2/) (/a(3),a(1),a(2)/)
```

などと配列の順序を替えた配列を簡単に作れる。これを使うと、ベクトルの外積は一行で書ける。

```
c(/1,2,3/)=a(/2,3,1/)*b(/3,1,2/)-a(/3,1,2/)*b(/2,3,1/)
```

良く使う配列の計算のために関数が用意されている。代表的なものを以下に挙げておく。

MAXVAL(a(:))	配列の最大値を求める。
MINVAL(a(:))	配列の最小値を求める。
PRODUCT(a(:))	全配列要素の積を求める。
SUM(a(:))	全配列要素の和を求める。
dot_product(a,b)	ベクトルの内積を求める。
mat_product(a,b)	ベクトルと行列の積を求める。

1.13 副プログラム

以前に組み込み関数の一覧を示したが、関数を自分で作ることができる。例えば、

```
program exfunc
   real(8)::x,y

   read(*,*) x
   y=SECOND(x*Pi/180.0)
   write(*,*) 'Second(',x,')=',y

   contains
   function SECOND(X) result(Y)
      real(8),intent(in)::X
      real(8):: Y
      Y=1.0/COS(X)
   end function SECOND

end program exfunc
```

1.14 ファイル入出力

これまで作成してきたプログラムでは、データの入出力は、

```
      read(*,*)   A
      write(*,*)   A
```

のように、キーボードから読み込み、ディスプレイに表示させるやり方で行なってきた。

つまり、

キーボード入力		ディスプレイ出力
	→ 計算 →	
(read(*,*) 文で)		(write(*,*) 文で)

のような流れになっている。

例えば、次の例 1 のプログラムでは、5 つの数値 (実数) を読み込みそれぞれの値を 3 倍してディスプレイに表示させることになる。

例 1.PQR.F90

```
program PQR
   implicit none
   real(8), dimension(5):: A,B
   integer I
!
   read(*,*) (A(I),I=1,5)
!
   do I=1,5
      B(I)=A(I)*3.0
   end do
!
   write(*,*) (B(I),I=1,5)
!
   do I=1,5
      write(*,*) B(I)
   end do
!
   end program PQR
```

例 1 のプログラムで、キーボードから数値データを入力する代わりにディスク上のファイルから数値を読み込ませたり、ディスプレイに結果を出力する代わりにファイルに結果を書き込んだりするにはどのようにすればよいであろうか。

例 1 で、XYZ.DAT(A ドライブ上) から変数 A(I) の値を読み込み、XYZ2.DAT に変数 B(I) の値を書き込むプログラムを考える。

流れとしては、

```
      入力                          出力
      from          →  計算   →     to
    A:XYZ.DAT                       A:XYZ2.DAT
```

のようなものにしたい。次に示す例 2 のプログラムは、例 1 のプログラムをこの流れになるように書き換えたものである。

例 2.XYZ1.F90

```
program XYZ1
   implicit none
   real(8),dimension(5):: A,B
   integer I

   open(1,file='A:XYZ.DAT')
   open(2,file='A:XYZ2.DAT')

   read(1,*) (A(I),I=1,5)
   do I=1,5
      B(I)=A(I)*3.0
   end do

   write(2,*) (B(I),I=1,5)
   do I=1,5
      write(2,*) B(I)
   end do
!
   close(1)
   close(2)
 end program XYZ1
```

例 2 の例 1 との違いは、

i) open 文、close 文が入っている。

ii) read(1,*), write(2,*) のように、read 文・write 文の中にファイル番号がある。

※ ファイル入出力を行うためには次の I ～III の要領でプログラミングすればよい。

I. read 文・write 文が出て来るより前に、open 文を書く。プログラムの冒頭の宣言文の直後に書くとよい。

例 2 では、

```
open(1,file=……)
```

```
open(2,file=……)
```

といった open 文が出てきている。

【書き方】　　open(番号, file=’ファイル名’)

- 番号は 1〜4、7〜9 の間の整数を使うこと
- ファイル名には、クオーテーションマーク’…’ を忘れないこと
- ファイル名はドライブ名 (例 2 では A:) を必ず頭につけること (そのファルがプログラムの存在するドライブにある場合でも必要)

II. read 文・write 文の中の番号 (これを「装置指定子」という：従来の端末入出力の場合は、“*” を指定していた) を 1〜4、7〜9 の間の任意の番号に替える。“5”、“6” は、端末入出力用 (すなわち “*” と同じ) ということに決まっているので使わないこと。

例 2 では、

```
read(1,*)
write(2,*)
```

となっている。

I の open 文の中の番号は、この read/write 文の中の番号 (装置指定子) に合わせること。open 文の中のファイル名は、read/write するファイル名を書く。

例 2 の open 文は、

```
open(1,FILE='A:XYZ.DAT')
open(2,FILE='A:XYZ2.DAT')
```

となっている。

read(1,*),write(2,*) となっているので、

read(データの入力) は、A ドライブ上の XYZ.DAT から、
write(データの出力) は、A ドライブ上の XYZ2.DAT にされる

わけである。

要するに、1、2 というのはファイル名をいちいち書く代わりに使われる指標である。

III. read/write 文の後に close 文を書く。close 文は、ファイル入出力を終了する命令なので、close 文が出てきた後に、例えば read(2,…) のような文が出てきてはいけない。従って、close 文はプログラムの一番最後、end 文の直前に書くのがよい。例 2 では、

```
close(1)
close(2)
```

となっている。

【書き方】　　close(番号)

【注意】 当然の話であるが、READ するファイル (例 2 では A:XYZ.DAT) は、プログラムを走らせる前に存在していなければならない。また、A(I) に対応する数値データもその中になければならない。

WRITE するファイル (例 2 では、A:XYZ2.DAT) は、予め存在していてもよいし、存在していなくてもよい。プログラムを走らせる段階で、A:XYZ2.DAT がなければ、自動的にこの名前のファイルが作られ、データが書き込まれる。もし、予め XYZ2.DAT が存在すれば、プログラムを走らせた時点でデータの内容は書き換えられる。

【注意】 READ するファイルと WRITE するファイルが同じであることは許されない。

【参考】 例 2 でててきた open/close 文は、指定子をできるだけ少なくした最も簡単な形 (指定子は、装置指定子とファイル名の 2 つだけしか与えていない) に書かれている。もっと厳密に書けば、

```
open(1,FILE='A:XYZ.DAT',STATUS='OLD',ACCESS='SEQUENTIAL',FORM='FORMATTED')
open(2,FILE='A:XYZ2.DAT',STATUS='NEW',ACCESS='SEQUENTIAL',FORM='FORMATTED')
```

となる。各指定子の意味は参考書を参照のこと。意味がわかれば、今日の授業でとりあげた以外のいろいろなファイル操作ができるようになるはずである。

まとめると例 2 の流れは、

```
入力  read(1,*)                出力 write(2,*)
from   →  計算    → to
A:XYZ.DAT              A:XYZ2.DAT
```

```
1 番  =  "A:XYZ.DAT"            2 番  =  "A:XYZ2.DAT"
  ↑                                ↑
      この指定を open ／ close 文で行う
```

当然 2 つ以上のファイルから読み込んだり、2 つ以上のファイルに出力させたりすることもできる。

実行例：

予め、エディターで、A ドライブ上に、XYZ.DAT を作成しておく。ファイルの中身は、

```
2.0  4.0  5.0  7.0  10.0
```

のように 5 つの値が入っている。

例 2 のプログラム (XYZ1.F90) を作成する。プログラムを走らせる前には、XYZ2.DAT はまだ存在していない。

プログラムの翻訳・結合 (LINK)・実行を行うと、

```
A>XYZ1

STOP - PROGRAM TERMINATED.
PRSS <RETURN> TO COMMAND MODE.
```

のように画面に表示され、XYZ2.DAT が自動的に作成される。

XYZ2.DAT の中身をエディターで見てみると、プログラムの write(2,*) 文で出力された数値データが書かれていることがわかる。

※問題 10　上の例を自分で実行させて、ファイル入出力ができることを確認せよ (XYZ.DAT は各自エディターでプログラム実行前に作成しておくこと)。(プログラム名　XYZ1.F90)

※問題 11　問題 10 で作ったプログラムを、ディスク上のファイルとの間で入出力ができるように変更せよ。但し、計算結果は 2 つのファイル (XYZ2.DAT および XYZ3.DAT) に出力させよ。作成したプログラムを実行させよ。

(プログラム名　MON11.F90)

さて、例 2 のようなプログラムを再び使って、今度は FGH.DAT なるファイルからデータを読ませるにはどうすればよいであろうか。当然、open 文を

```
open(1,FILE='A:FGH.DAT')
```

のように書き換えればよい。いま、そのように書き換えたプログラムの名前を仮に FGH.F90　とする。

しかし、同じたぐいの数値処理を行うのにもかかわらず、一旦作ったプログラムを書き換えて翻訳・結合・実行の手順を一からやり直すのはいかにも面倒であるし、それでは、1 つの入力ファイルにつき 1 つのプログラムが必要なことになる (つまり、入力ファイルが XYZ.DAT のときは XYZ.F90 を走らせ、入力ファイルが FGH.DAT のときには FGH.F90 を走らせることになり、同じようなプログラムが 2 つできてしまう)。

そこで、下の例 3 のようにプログラムを書いておけば、実行する段階でその都度入力ファイル名を自由に指定できることになり、便利である。例 2 のようなオーソドックスなファイル入出力のやり方のパターンに慣れれば、例 3 のような方法を使うことを勧める。

例 3.XYZ.F90

```
program XYZ2
   implicit none
   real(8)::A(5),B(5)
   character(15)::IFLI
!
   write(*,*) 'INPUT FILE NAME FOR INITIAL DATA'
   read(*,'A15') IFLI
!
   open(1,FILE=IFLI)
   open(2,FILE='A:XYZ2.DAT')
!
   read(1,*) (A(I),I=1,5)
```

```
   do I=1,5
      B(I)=A(I)*3.0
   end do
!
   write(2,*) (B(I),I=1,5)
!
   do I=1,5
      write(2,*) B(I)
   end do
!
   close(1)
   close(2)
end program XYZ2
```

【参考】 このプログラムで、character(15)::IFLI とは、変数 IFLI は、数値ではなく、文字列 15 文字をデータに持っているという意味である。(普通、変数は数値をデータとして持っている。例えば、A=1.0 というステートメントがあれば、変数 A は数値 1.0 をデータとして持っているわけである。例 3 の場合は IFLI は数値でなく、例えば'A:FGH.DAT' といった文字列をデータに持つことができるわけである。

read(*,'A15') IFLI とは、変数 (文字列)IFLI をキーボード (5 番) から、FORMAT 'A15' という書式で読み込めという意味である。FORMAT 'A15' は、文字型データ 15 文字という書式を指定する。

以上何のことか意味がわからないかもしれないが、今は鵜呑みでよい。後ほど解説する。

例 3 のプログラムを実行させた例は以下の通り。

```
A>XYZ2

INPUT FILE NAME FOR INITIAL DATA
A:FGH.DAT                    ←  入力ファイル名をキーボードから入れる

STOP - PROGRAM TERMINATED.
PRESS <RETURN> TO COMMAND MODE.

A>
```

このように、

```
INPUT FILE NAME FOR INITIAL DATA
```

と画面に表示された後、入力待ちの状態になっているので、ここで、入力ファイル名を入れてやれば、そのファイルからデータを読み取って実行される。

※問題 12　例 3 のプログラムをさらに、write(2,*) 文の出力ファイル名も実行時にキーボード入力で指定できるように変更せよ。

答)

```
program XYZ
   implicit none
   real(8):: A(5),B(5)
   character(15)IFLI,OFLI
!
   write(*,*) 'INPUT FILE NAME FOR INITIAL DATA'
   read(*,'A15') IFLI
   write(*,*) 'OUTPUT FILE NAME FOR FINAL DATA'
   read(*,'A15') OFLI
!
   open(1,FILE=IFLI)
   open(2,FILE=OFLI)
!
      ＜以下同じ＞
```

1.15　書式の指定について

これまで作成してきたプログラムでは、データの入出力は、

```
read(*,*) A
write(*,*) A
```

のように書かれていた。

実は、READ ／ WRITE 文の括弧の中の “*” は、書式の設定をしないという意味だったのだ。書式とは、例えば、

```
A=12345.678
```

と定義された変数 A の値を (ディスプレイやファイルに) 出力させるとき、

1.23E4	と表示させるか
12345.6	まで表示させるか
12345.67	まで表示させるか、

といった表示のさせ方のことである。これは単に表示のさせ方だけの問題である (A の数値自体は不変である) ! 書式を指定するためには、FORMAT 指定を使う。FORMAT 指定は必ず WRITE 文や READ 文の第二引き数に文字列として入れる。書き方は、例えば、

```
write(*,'……') A
```

のように、WRITE 文の括弧の中で書式の指定をする。すると’……’ の中の指定に応じて、A の値が、1.23E4 と出力されたり、12345.67 と出力されたりすることになる。

FORMAT 指定では、出力する変数が、整数型か実数型か、あるいは文字型かによって指定の書き方 (記号) が違っている。とりあえず以下の例を見てほしい。

例 1. 整数型変数の出力

```
program SY01
   implicit none
   J=123456789
   K=987654321
   write(*,*)   J,K
   write(*,’3X,I4’) J
   write(*,’3X,I9’) J
   write(*,’6X,I9’) J
   write(*,’3X,I12’) J
   write(*,’3X,2I9’) J,K
   write(*,’3X,I9,I9’) J,K
   write(*,’3X,I9,3X,I9’) J,K
   write(*,’3X,2(I9,3X)’) J,K
end program SY01
```

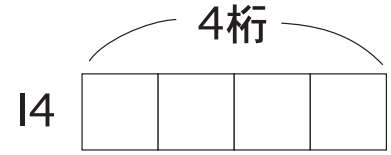
例 1 の例えば 2 番目のの FORMAT 指定に出てきた “3X” というのは、空白を 3 カラム文あけよという意味で、I9 は整数を 9 桁で表示させよという意味である。この結果、

```
write(*,’3X,I9’)   J
```

では、空白が 3 カラム出力された後に (同じ行に)、J の値が 9 桁で表示されることになる。

整数型変数を出力させるには、このような「I 型編集記述子」を使って指定する。

例えば、I4 と指定すると、次のようになる。



最後の’2(I9,3X)’ は、’I9,3X,I9,3X’ と同じ意味である。つまり、「はじめの数値 (J) が 9 桁で表示された後、空白 3 カラム、次の数値 (K) が 9 桁で表示、空白 3 カラム」という書式で出力される。

つまり、

```
nIw
```

n は使用する I 型編集子の個数、w は桁数。

例 2. 実数型変数の出力

```
program SY02
   implicit none
   P=12345.67
   Q=67890.12
   write(*,*)   P,Q
   write(*,'3X,F4.0') P
   write(*,'3X,F6.0') P
   write(*,'6X,F8.2') P
   write(*,'3X,E4.0') P
   write(*,'3X,E6.0') P
   write(*,'3X,E15.7') P
   write(*,'3X,2(F21.10,2X)') P,Q
   write(*,'3X,2(E21.10,2X)') P,Q
end program SY02
```

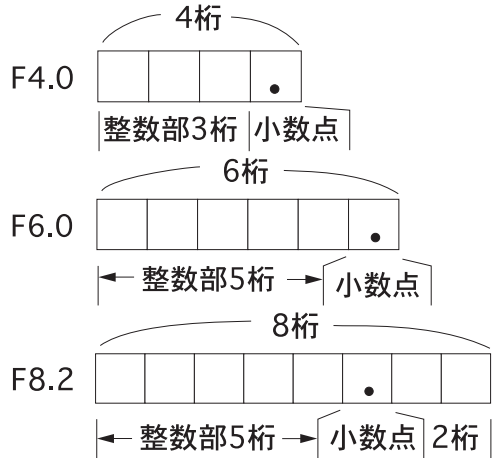
実数型変数を出力させるには、「F 型編集記述子」や「E 型編集記述子」を使って指定する。E 型は、出力結果を指数表示で行うもので、F 型は単なる小数として表示するものである。

F 型編集記述子は次のように編集を行う。

```
nFw.d
```

n は使用する F 型編集子の個数、w は全桁数、d は小数部の桁数。小数点も桁数に含まれるので、整数部の桁数は w−1−d になる。

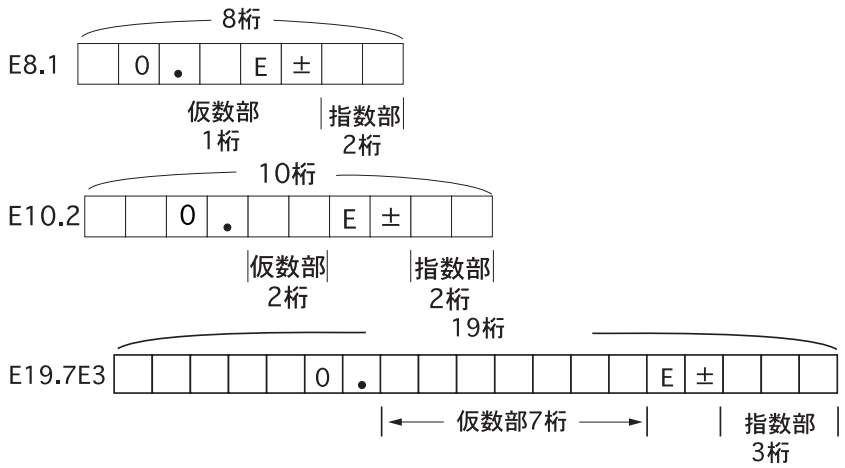
例えば、



E 型編集記述子は次のように編集を行う。

```
nEw.d
```

n は E 型編集子の個数、w は全桁数、d は小数の桁数



例 3. 文字型変数の出力

```
program SYO3
   implicit none
   CHARACTER(15)::IJK,PQR
   IJK='ABCDEFGHIJ'
   PQR='0123456789'
   write(*,'3X,A6') IJK
   write(*,'3X,A10') IJK
   write(*,'3X,A10,2X,A10') IJK,PQR
   write(*,'3X,A12,2X,A12') IJK,PQR
end
```

ある変数が、「文字型」であることを宣言するには、CHARACTER 文を使う。例 3 に示すとおり、

```
CHARACTER(n)::IJK
```

は変数 IJK に N 個の文字数をもつ文字列が代入されることを宣言する。例 3 では実際には、IJK には 10 個の文字数をもつ文字列'ABCDEFGHIJ' が代入されている。このような文字型変数を実際にどのように使うかという具体例は、配布資料 4 の例 3 を参照。

文字型変数を出力させるには、「A 型編集記述子」を使って指定する。

A 型編集記述子は次のように編集を行う。

nAw

n は使用する A 型編集子の個数、w は出力する文字数。

例えば、

例 4. 任意の文字列の出力

```
program SY04
   implicit none
   write(*,*)  'ABCDEFGHIJ'
   write(*,'3X,"ABCDEFGHIJ")
   A=1.0
   write(*,'3X,"A=",2X,E12.5') A
   write(*,'3X,A4,E12.5') 'A=  ',A
end program SY04
```

任意の文字列を出力させるには例 4 のような方法がある。

※問題 28　例 1〜4 のプログラムを作成し、実行させ、どのように出力されるかを見よ。出力結果の画面コピーをとれ。

****の表示は、まともに数値の表示ができなかったことを意味している (変数の桁数より FORMAT 指定した桁数が少なかった、など)。その原因を考えよ。

READ 文に対しても、同様の書式指定ができる。但し、数値を入力する際には、指定した書式で入力させなければならない。

実際上、文字型変数の入力以外は、`read(*,*)` のように書式指定なしで READ 文を書いておいた方が、どのような書式の数値データでも入力できて便利である。

ディスク上のファイルなどに WRITE 文で出力させた数値データを、別のプログラムで READ 文で読み込む場合、出力時の WRITE 文の書式と、入力時の READ 文の書式は同じにしておいた方が (初心者は) 無難である。特に、入力データよりも少ない桁数の書式で読み込んだ場合、入力された数値の正しさは保証されないので注意が必要である。これを避けるため、例えば、

```
do I=1,N
   write(1,'2X,E12.5,2X,E12.5') A(I),B(I)
end do
```

で出力された数値 `A(I),B(I)` を読み込む場合は、

```
do I=1,N
   read(1,'2X,E12.5,2X,E12.5') A(I),B(I)
end do
```

のように同じ書式で読み込むのが無難。もっともこの場合は、`read(1,*) A(I),B(I)` で読み込んでも問題は生じない。

ここでとりあげた FORMAT の指定は、数ある書式指定の中のごく一部である。詳しくは、Fortran90 の文法書を参照されたい。

これまでに

I. 整数と実数の区別

II. ステートメントの意味 (右辺の値を左辺に代入するということ、組込み関数

III. 繰り返し計算の方法 (do 文、配列変数)

IV. 条件判断・分岐の方法 (IF/GO TO 文)

V. データの入出力の方法 (READ/WRITE/open/close/FORMAT)

を習得した。実はこれで科学技術計算に必要最小限の文法はマスターしたのだ！以上で、Fortran90 の解説を終了する。これで、Fortran90 を使って必要最小限のプログラミングはできるようになった (はずである)。科学技術計算をするための最短コースを短時間で走ってきたので、多くの詳細を省略したし、科学技術計算の経験上、最も有用と思われる 1 つのやり方を勧めてきたに過ぎない。例えば、まだ他に、サブルーチン (副プログラム)、全く取り上げていない事柄もあるし、宣言文、条件文や do 文についてももっといろいろな使い方があるが実習では取り上げなかった。それらについては、Fortran90 の参考書を見て各自で補ってもらいたい。次章では、科学技術計算の初歩についてとりあげる (例えば、データ解析や微分や積分をコンピューターで行うにはどうすればよいか)。

第2章　練習問題

※問題 13　関数　$y = \sin x$ において、20 個の x の値 $x_n = n\pi/10 (n = 1, 2, \cdots, 20)$ に対する y の値,y_n, をそれぞれ求め、(x_n,y_n) の数値の組 (20 点) を、ディスクに出力させるプログラムを作成し、実行させよ。(プログラム名　MON13.F90)
(ヒント：$\sin x$ は Fortran90 では　SIN(X) と書く。do 文を使うこと。できるだけ配列変数を使うこと。)

※問題 14　問題 13 でディスクに作成したデータを読み込み、$y_n (n = 1, 2, \cdots, 20)$ をそれぞれ $\pi/10$ 倍した値の総和を求め、結果をディスプレーに表示させるプログラムを作成せよ。プログラムを実行させよ (結果はいくらか？)。(プログラム名　MON14.F90)
(要するに、$S = \sum_{n=1}^{20} y_n \cdot \pi/10$ を求める)

以上 Fortran 文法に関する練習問題 (問題 15〜問題 27)

※問題 15　級数 $\sum_{k=0}^{n} (2k+1)^2$ を求めるプログラムを作成せよ。(プログラム名 MON15.F90)

※問題 16　乗積 $\prod_{k=2}^{n} (k-1)^2$ を求めるプログラムを作成せよ。(プログラム名 MON16.F90)

※問題 17　4 次式

$$y = 1.3x^4 - 0.9x^3 + 6.3x^2 + 0.4x + 2.2$$

の値を、0.1 から 0.2 刻みで 3.4 までの x に対して計算し、x と y を出力するプログラムを作成せよ。(プログラム名 MON17.F90)

※問題 18　任意の自然数 n に対し、$(2n-1)!! = (2n-1)(2n-3) \cdot \cdots \cdot 3 \cdot 1$ を計算するプログラムを作成せよ。(プログラム名 MON18.F90)

※問題 19　指数関数の Maclaurin 展開は、

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

である。上の展開式の右辺の無限級数を第 m 項まで求めることによって、$e^{0.1}$ を近似的に求めるプログラムを作成せよ。Fortran の組み込み関数を使えば e^x は EXP(X) と書けば求められる。級数の和として得られた値を、組み込み関数を使って得られた値と比較せよ。(プログラム名 MON19.F90)

※問題 20　次のような関係が知られている。

$$\prod_{r=1}^{n-1} \sin \frac{r\pi}{n} \left(= \sin \frac{\pi}{n} \sin \frac{2\pi}{n} \cdots \sin \frac{(n-1)\pi}{n}\right) = \frac{n}{2^{n-1}}$$

左辺の乗積を求めるプログラムを作り、右辺の値と比較せよ。(プログラム名 MON20.F90)

※問題 21　3 次元空間の 2 つの点の座標 (x_1, y_1, z_1),(x_2, y_2, z_2) を読み込み、2 点間の距離を求めるプログラムを作れ。(プログラム名 MON21.F90)

※問題 22　3 角形の 3 辺の長さ A,B,C を読み込み、ヘロンの公式によって面積を求めるプログラムを作成せよ。

(ヘロンの公式：

$$S = \{w(w-a)(w-b)(w-c)\}^{\frac{1}{2}}, \qquad w = \frac{1}{2}(a+b+c)$$

(プログラム名 MON22.F90)

※問題 23　フィボナッチ数列の最初の 20 項を求めて出力するプログラムを作成せよ。

(フィボナッチ数列とは、最初の 2 項が $f_1 = 0, f_2 = 1$ で始まり、以降は順次、反復公式

$$f_n = f_{n-2} + f_{n-1}$$

を用いて定められる数列のことである。) (プログラム名 MON23.F90)

※問題 24　n 個の数値 (実数) を読み込んで、その算術平均と幾何平均を求めるプログラムを作成せよ。(プログラム名 MON24.F90)

※問題 25　n 個の整数を読み込ませて、その中の最大値と最小値をみつけるプログラムを、IF 文を使って書け。(プログラム名 MON25.F90)

※問題 26　Pa(パスカル) 単位で表された圧力を読み込んで、GPa,MPa,KPa,Pa で表現して出力するプログラムを作成せよ。(G(ギガ)：10^9；　M(メガ)：10^6) (プログラム名 MON26.F90)

2.1　応用問題

2.1.1　最小 2 乗法 (1)

一組の変量 (x, y) について n 個の観測点 $(x_i, y_i)(I = 1, 2, \cdots, n)$ があるとき、これらの観測点を最もよく表していると思われる関数 $y(x)$ をつくることを考える。

(A) 原点を通る 1 次関数

n 個の観測点 $(x_i, y_i)(I = 1, 2, \cdots, n)$ を原点を通る 1 次関数

$$y = a_1 x \tag{2.1}$$

で近似することを考える。

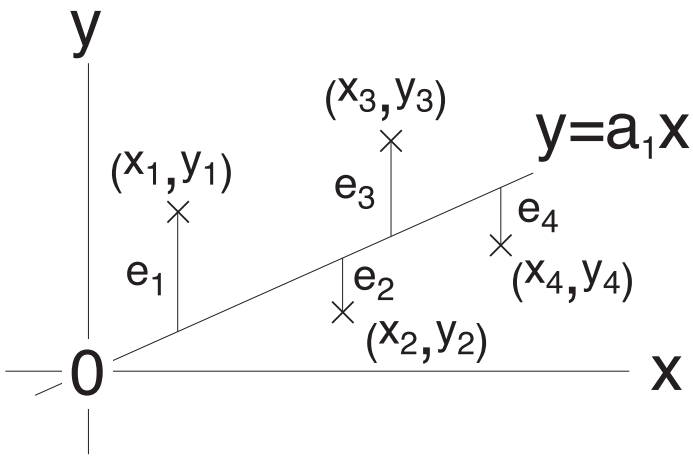
(2.1) 式と観測値 y_1 との差 (残差) を e_1 すると

$$e_i = y_1 - a_1 x_i \tag{2.2}$$

となる。ここで残差の 2 乗和を f とすると、

$$f = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \{y_i - a_1 x_i\}^2 \tag{2.3}$$

となる。f を最小にするような a_1 を決定すれば、その直線が与えられた観測点のデータを最もよく表している直線であると考えることができる (下図参照)。



(2.3) 式を a_1 について微分すると、

$$\frac{\partial f}{\partial a_1} = -2\sum_{i=1}^{n}(x_i y_i - a_1 x_i^2) = 0 \tag{2.4}$$

(極値の条件) となる。

これから

$$\sum_{i=1}^{n} x_i y_i = a_1 \sum_{i=1}^{n} x_i^2 \qquad \text{故に } a_1 = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} \tag{2.5}$$

となる。観測データの最もよく合う原点を通る 1 次関数の傾きをこのようにして決めることができる。このような方法を最小自乗法という。

(B) 原点を通らない 1 次関数 n 個の観測点 $(x_i, y_i)(i = 1, 2, \cdots, n)$ を原点を通らない 1 次関数

$$y = a_0 + a_1 x \tag{2.6}$$

で近似することを考える。

(2.6) 式と観測値 y_i との差 (残差)e_i は、

$$e_i = y_i - (a_0 + a_1 x_i) \tag{2.7}$$

となる。残差の 2 乗和 f は、

$$f = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}\{y_i - (a_0 + a_1 x_i)\}^2 \tag{2.8}$$

となる。f を最小にするような a_0,a_1 は、f の a_0,a_1 のよる偏微分がゼロという条件から定まる。すなわち、

$$\frac{\partial f}{\partial a_0} = -2\sum_{i=1}^{n}\{y_i - (a_0 + a_1 x_i)\} = 0$$

$$\frac{\partial f}{\partial a_1} = -2\sum_{i=1}^{n}\{(y_i - (a_0 + a_1 x_i))x_i\} = 0$$

すなわち、

$$\sum_{i=1}^{n} y_i = a_0 \sum_{i=1}^{n} 1 + a_1 \sum_{i=1}^{n} x_i \tag{2.9}$$

$$\sum_{i=1}^{n} y_i x_i = a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2$$

ここで、

$$\sum_{i=1}^{n} y_i = T_0, \qquad \sum_{i=1}^{n} y_i x_i = T_1, \qquad \sum_{i=1}^{n} 1 = S_0(= n), \qquad \sum_{i=1}^{n} x_i = S_1, \qquad \sum_{i=1}^{n} x_i^2 = S_2 \tag{2.10}$$

とおけば、(2.9) 式は

$$\begin{pmatrix} S_0 & S_1 \\ S_1 & S_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} T_0 \\ T_1 \end{pmatrix}$$

$$a_0 = \frac{T_0 S_2 - T_1 S_1}{n S_2 - S_1^2}, \qquad a_1 = \frac{n T_1 - T_0 S_1}{n S_2 - S_1^2} \tag{2.11}$$

となる。

(C)m 次関数

n 個の観測点 $(x_i, y_i)(i = 1, 2, \cdots, n)$ を m 次関数

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m \tag{2.12}$$

で近似することを考える。

(2.12) 式と観測値 y_i との差 (残差)e_i は、

$$e_i = y_i - \sum_{j=0}^{m} a_j {x_i}^j \tag{2.13}$$

となる。残差の 2 乗和 f は、

$$f = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left\{ y_i - \sum_{j=0}^{m} a_j {z_i}^j \right\}^2 \tag{2.14}$$

となる。f の a_k による偏微分がゼロという条件は、

$$\frac{\partial f}{\partial a_k} = -2\sum_{i=1}^{n} \left[\left\{ y_i - \sum_{j=0}^{m} a_j {x_i}^j \right\} {x_i}^k \right] = 0, \quad (k = 0, 1, \cdots, m)$$

すなわち

$$\sum_{i=1}^{n} y_i {x_i}^k = a_0 \sum_{i=1}^{n} {x_i}^k + a_1 \sum_{i=1}^{n} {x_i}^{k+1} + \cdots + a_m \sum_{i=1}^{n} {x_i}^{k+m} \tag{2.15}$$

ここで、

$$\sum_{i=1}^{n} y_i {x_i}^k = T_k, \qquad \sum_{i=1}^{n} {x_i}^k = S_k \tag{2.16}$$

とおけば、(2.15) 式は

$$\sum_{j=0}^{m} a_j S_{k+j} = T_k, \quad (k = 0, 1, \cdots, m) \tag{2.17}$$

となる。これは $(m+1)$ 元の連立 1 次方程式

$$\begin{pmatrix} S_0 & S_1 & S_2 & \cdots & S_m \\ S_1 & S_2 & S_3 & \cdots & S_{m+1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ S_m & S_{m+1} & S_{m+2} & \cdots & S_{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_m \end{pmatrix} = \begin{pmatrix} T_0 \\ T_1 \\ \cdot \\ \cdot \\ \cdot \\ T_m \end{pmatrix} \tag{2.18}$$

である。これを解けば、(2.12) 式の各項の係数が求められる。

従って、われわれは 2 次式への最小 2 乗フィッティングは行うことができるはずである。

今後各種の実験データの処理に際しては、目測によって直線 (ないし曲線) をひくのではなく、最小 2 乗法によって線をひくこと。

※問題 29　ディスク上の N 個のデータ $(x_i, y_i)(i = 1, 2, \cdots, N)$ のデータを読み込み、(原点を通らない)1 次関数に最小 2 乗法でフィッティングするプログラムを作成せよ。ファイル入出力のやり方は 5 節を参照のこと。

【プログラム例】

```
program MON29
   implicit none
   real(8),dimension(100)::X,Y
   real(8)::XSum,YSum,XAv,YAv,SX,SXY,A,B
   character(15)::IFLI
   write(*,*) 'INPUT FILE NAME FOR DATA'
   read(*,'A15') IFLI
!
   open(1,FILE=IFLI)
!
   read(1,*) N
   PN=FLOAT(N)
!
   do I=1,N
      read(1,*) X(I),Y(I)
   end do
```

```
   Xs=SUM(X(1:N))
   Ys=SUM(Y(1:N))
   XAV=XSUM/PN
   YAV=YSUM/PN
   Sxx=SUM((X(1:N)-Xav)**2)
   Sxy=SUM((X(1:N)-Xav)*(Y(1:N)-Yav))
   A=Sxy/Sxx
   B=Yav-A*Xav
   write(*,301) A,B
  301 FORMAT(3X,'A = ',E12.5,3X,'B = ',E12.5)
   close(1)
end mon29
```

※問題 30　問題 29 で作成したプログラムを用いて、次の表のデータを直線で近似せよ。まず、次表の数値をディスク上のファイルに (エディターを使って) 書き、数値をプログラムを走らせるときに読み込ませよ。

※問題 31　ディスク上の N 個のデータ $(x_i, y_i)(i = 1, 2, \cdots, N)$ のデータを読み込み、2 次関数に最小 2 乗法でフィッティングするプログラムを作成せよ。

※問題 32　問題 30 のデータを問題 31 で作成したプログラムを用いて、2 次関数で近似せよ。

※問題 33　問題 31 で作成したプログラムを用いて、次の表のデータを 2 次関数で近似せよ。まず、次表の数値をディスク上のファイルに (エディターを使って) 書き、数値をプログラムを走らせるときに読み込ませよ。

x	0	100	200	300	400	500	600	700	800	900	1000
y	8.24	9.40	10.70	12.15	13.40	14.60	15.65	16.60	17.40	18.23	18.93

2.1.2　最小 2 乗法 (2)

n 個の観測点 $(x_i, y_i)(i = 1, 2, \cdots, n)$ を、m 次関数

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m \tag{2.19}$$

へ最小 2 乗法によってフィッティングをすると、各項の最適化された係数は、連立一次方程式

$$\begin{pmatrix} S_0 & S_1 & S_2 & \cdots & S_m \\ S_1 & S_2 & S_3 & \cdots & S_{m+1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ S_m & S_{m+1} & S_{m+2} & \cdots & S_{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_m \end{pmatrix} = \begin{pmatrix} T_0 \\ T_1 \\ \cdot \\ \cdot \\ \cdot \\ T_m \end{pmatrix} \tag{2.20}$$

の解 $a_0, \cdots, a_m$ として与えられる。ここで、

$$S_k = \sum_{i=1}^{n} x_i^k \tag{2.21}$$

$$T_k = \sum_{i=1}^{n} y_i x_i^k$$

である。

一般に、m 元連立一次方程式

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_m \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{pmatrix} \tag{2.22}$$

の解 $x_1, x_2, \cdots, x_m$ は、クラメルの公式

$$x_j = \frac{\begin{vmatrix} a_{11} & \cdots & a_{1j-1} & b_1 & a_{1j+1} & \cdots & a_{1m} \\ a_{21} & \cdots & a_{2j-1} & b_2 & a_{2j+1} & \cdots & a_{2m} \\ \cdot & & \cdot & \cdot & \cdot & & \cdot \\ a_{m1} & \cdots & a_{mj-1} & b_n & a_{mj+1} & \cdots & a_{mm} \end{vmatrix}}{\begin{vmatrix} a_{11} & \cdots & a_{1j-1} & a_{1j} & a_{1j+1} & \cdots & a_{1m} \\ a_{21} & \cdots & a_{2j-1} & a_{2j} & a_{2j+1} & \cdots & a_{2m} \\ \cdot & & \cdot & \cdot & \cdot & & \cdot \\ a_{m1} & \cdots & a_{mj-1} & a_{nj} & a_{mj+1} & \cdots & a_{mm} \end{vmatrix}} \qquad (j = 1, 2, \cdots, m) \tag{2.23}$$

によって与えられる。この公式を使えば、(2.20) 式は解ける (m 次関数へのフィットは $(m+1)$ 次の行列式を使うことになる)。ただし、クラメルの公式は 4 次の行列式くらいが実際にプログラミングできる限界であろう。それより多元の連立 1 次方程式の解は、行列式を求めるサブルーチンを使って解くべきである。行列式を求める効率的なアルゴリズムについては成書を参照されたい。

例：

(1)1 次関数

$$y = a_0 + a_1 x$$

へのフィットは、連立一次方程式

$$\begin{pmatrix} S_0 & S_1 \\ S_1 & S_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} T_0 \\ T_1 \end{pmatrix}$$

を解けばよい。解はクラメルの公式

$$a_0 = \frac{\begin{vmatrix} T_0 & S_1 \\ T_1 & S_2 \end{vmatrix}}{\begin{vmatrix} S_0 & S_1 \\ S_1 & S_2 \end{vmatrix}}, \qquad a_1 = \frac{\begin{vmatrix} S_0 & T_0 \\ S_1 & T_1 \end{vmatrix}}{\begin{vmatrix} S_0 & S_1 \\ S_1 & S_2 \end{vmatrix}}$$

によって与えられる。

【プログラム例】

```
program MON29A
   implicit none
   real(8),dimension(100)::X,Y
   real(8)::RN,Sx,Sx2,Sy,Sy2,DD,A0,A1
   integer::N
   character(15)::IFLI
   write(*,*) '   INPUT FILE NAME FOR DATA')
   read(*,'A15') IFLI
!
   open(1,FILE=IFLI)
!
   read(1,*) N
   RN=FLOAT(N)
!
   do I=1,N
      read(1,*) X(I),Y(I)
   end do
!
   Sx =SUM(X(1:N))
   Sx2=SUM(X(1:N)**2)
   Sy =SUM(Y(1:N))
   Sy2=SUM(Y(1:N)**2)
!
   DD=RN*Sx2-Sx**2
   A0=(Sy*Sx2-Sy2*Sx)/DD
   A1=(RN*Sy2-Sx*Sy)/DD
!
   write(*,'   y=',E12.5,'+',E12.5,'x') A0,A1
   close(1)
end program MON29
```

2.2　地球科学的応用問題

- 緯度経度で与えられた球面上の一点を表すベクトルを計算する。

$$
\begin{aligned}
x &= \cos\phi\cos\lambda \\
y &= \sin\phi\cos\lambda \\
z &= \sin\lambda
\end{aligned}
$$

- 与えられたベクトルの方位を緯度経度で表すプログラム。

$$\begin{aligned} \lambda &= \sin^{-1} z \\ \phi &= \tan^{-1} \frac{y}{x} \quad \text{ATAN2(y, x) を使うと良い} \\ & \quad \text{但し、} x^2 + y^2 + z^2 = 1 \text{ の場合} \end{aligned}$$

- 地球上の 2 点間の距離を計算する。

$$\cos \delta = P_1 \cdot P_2$$

- 球面上の 2 点を通る大円の極を計算する。

$$C = P_1 \times P_2$$

- 二つの大円（極の位置で与える）の交点。

$$P = C_1 \times C_2$$

- 二つの結晶面のなす角度 (面指数から面角を求める)。

  ヒント: 二つの面のなす角度は、面の垂線のなす角度である。

- 二つの地層面の交わる直線の方位 (しばしば、褶曲のプランジ軸を与える)。

  ヒント: 走向傾斜から垂線ベクトルを計算すれば垂線ベクトルの外積が交わる方位。

2.2.1 地図

地図データは gnuplot のディレクトリの world.dat にある。このデータは各大陸や島の海岸線上の点の緯度経度が順番に各行に入っている。一周して、次の大陸や島に移るところで一行空行が入れてある。これを gnuplot で beginverbatim gnuplot¿plot "world.dat" with line endverbatim とすると、正距円筒図法の地図が書ける。このデータを用いて、以下の投影法での海岸線の位置を計算するプログラムを作り、同様のファイルを作って、gnuplot で地図を書け。

- 正距円筒図法（緯度方向の距離が赤道での経度の距離に正しい）

$$\begin{aligned} x &= R\phi \frac{\pi}{180} \\ y &= R\lambda \frac{\pi}{180} \end{aligned}$$

- 正射円筒図法 (正積円筒図法)

$$\begin{aligned} x &= R\phi \frac{\pi}{180} \\ y &= R \sin \phi \end{aligned}$$

- 等角円筒図法（メルカトル図法）

$$\begin{aligned} x &= R\phi\frac{\pi}{180} \\ y &= R\log\left\{\tan\left(\frac{\pi}{4}-\frac{\phi}{2}\right)\right\} \end{aligned}$$

- 正距方位図法

$$\begin{aligned} r &= R\delta\frac{\pi}{180} \\ \theta &= \phi \end{aligned}$$

- 正射図法

$$\begin{aligned} r &= R\sin\delta \\ \theta &= \phi \end{aligned}$$

- 平射図法（ステレオ図法＝等角）

$$\begin{aligned} r &= 2R\tan\frac{\delta}{2} \\ \theta &= \phi \end{aligned}$$

- ランベルト正積方位図法

$$\begin{aligned} r &= 2R\sin\frac{\delta}{2} \\ \theta &= \phi \end{aligned}$$

2.2.2　プレート運動

- プレートの相対運動の極と角速度が与えられた時、境界上のとある点での相対速度の絶対値。

$$|V| = V_0\sin\delta$$

- プレート A から見た B の角速度、B から見た C の角速度が与えられた時、プレート A から見た C の角速度

$${}_AV_C = {}_AV_B + {}_BV_A$$

- プレートの相対運動の極と角速度が与えられた時、境界上のとある点での相対速度の方位と大きさ。

$$Az_v = E \times P \quad (\text{この後ローカル座標に変換する必要がある})$$

- プレートの相対運動の極と回転角が与えられた時、プレート上のとある点の回転後の位置。前期の地図の一部を使って、大陸の回転運動 (大陸移動の復元!!) の地図を作れると面白い。
- プレート A から見た B の過去 10my の回転運動、プレート B から見た C の過去 10my の回転運動、が与えられた時、プレート A から見た C の過去 10my の回転運動を求める。
- プレート A から見た B の過去 10my と 20my の回転運動が与えられた時、20Ma から 10Ma までの回転運動を求める。